PROJET :

CONCEPTION AGILE DE PROJETS INFORMATIQUES ET GENIE LOGICIEL

Réalisé par :

Yasmine Bendjeddou
Dieng Ndeye-Fatou
Bourhani Dounya

Professeur :
Lachand-Pascal Valentin

Introduction :

Dans le cadre de notre formation master 1 informatique il nous a été demandé de réaliser un projet dans un module nommé « Conception Agile de projet ».

Il nous a été demandé de mettre en place quelques classes pour nous remettre en tête les grands principes de la programmation orientée objet : messages et collaboration entre objets, attributs et méthodes, constructeurs, héritage, etc.

Pour cela, nous avons travaillé sur un jeu nommé balle aux prisonniers.

Le but du jeu est de réussir à toucher les joueurs de l'équipe adverse avec le ballon ce qui fera, du joueur touché un prisonnier.

Nous avons donc eu à modifier, créer des joueurs, des projectiles et implémenter des stratégies de tirs et des modèles de designs patterns.

Dans ce rapport, nous allons présenter le fonctionnement du jeu et détailler les designs patterns que nous avons utilisés durant ce projet.

I- Conception du jeu

1. Cas d'utilisation

Dans notre jeu nous avons créé 2 joueurs humains (rouge et bleu), 2 joueurs IA (contrôlés par l'ordinateur) et 2 Skeltons qui sont aussi des joueurs intelligents.

- Le joueur rouge humain se déplace grâce au bouton « **LEFT** » et « **RIGTH** ».
- Le joueur bleu humain se déplace grâce au touche **Q** et **D** (**Q** pour se déplacer vers la gauche et **D** pour se déplacer vers la droite.
- La balle du joueur rouge humain se déplace selon les directions qu'il veut avec les flèches du haut et du bas du clavier.
- La balle du joueur bleu humain se déplace selon les directions qu'il veut avec les touches **S** et **Z**
- Le joueur bleu tire avec la touche « **ESPACE** » et le joueur rouge tire avec la touche « **ENTER** »

Concernant les joueurs IA, nous n'avons pas réussi à les faire déplacer aléatoirement selon les déplacements du joueur humain de leurs équipes respectives.

2. Les classes :

Le projet comporte plusieurs classes :

➢ La classe App

La classe App permet l'exécution du jeu, c'est la classe principale de l'application qui s'appuie sur javafx

➢ La classe projectile

Cette classe projectile initialise les projectiles.

➢ La classe Player

Cette classe permet de gérer les joueurs. Elle permet l'affichage des équipes et des flèches lors de l'appel de la classe. Chaque équipe est constituée de 3 joueurs (un humain et deux joueurs IA).

➢ La classe Player IA

Cette classe permet de créer des joueurs intelligents.

➢ La classe Human Player

Cette classe permet de créer des joueurs humains, c'est-à-dire contrôlés par l'utilisateur du jeu.

- **La classe Field**

  Cette classe sert à gérer le terrain de jeu, on y trouve la gestion des projectiles, les événements claviers et souris et les animations.

- **La classe PlayerFactoryIF**

  Elle gère l'interface, elle est utilisée pour le design pattern factory.

- **La classe Sprite**

  Cette classe gère la balle.

- **La classe PlayerFactory**

  Elle est également pour le design pattern factory.

II- Design patterns

Nous avons décidé pour nos design patterns de partir tout d'abord sur le **Factory**.
C'est un patron de conception qui permet de créer des familles d'objets apparentés sans préciser leur classe concrète.
En effet, ce design pattern nous paraît utile pour la création de nos joueurs. Nous avons différencié certains comportements pour des joueurs dits « humains » (de la classe *HumanPlayer*) et des joueurs IA (de la classe *IAPlayer*).
Ce design pattern nous permet de créer une instance de *Player* à partir de la classe dérivée (*HumanPlayer* ou *IAPlayer*), selon un critère donné et en nous évitant de systématiquement devoir évaluer le critère de choix.
La classe exacte utilisée pour produire l'objet n'est donc pas connue par l'appelant, ici le constructeur de la classe *Field*.

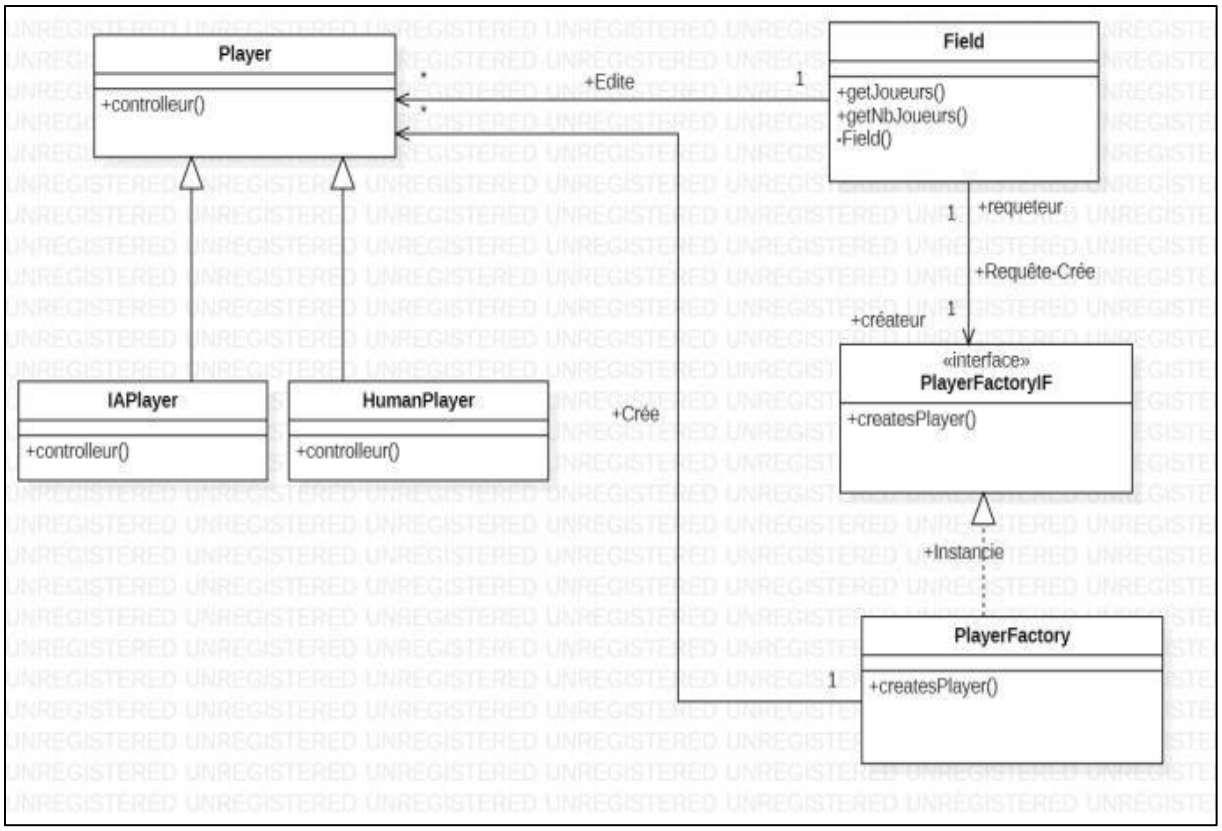

Figure 1: Diagramme de Factory

Ensuite, nous nous sommes penchées sur le design pattern **Singleton**.
C'est un patron de conception de création qui garantit que l'instance d'une classe n'existe qu'en un seul exemplaire, tout en fournissant un point d'accès global à cette instance.
Nous l'avons mis en place sur la classe *Field* car pour le jeu nous avons en effet besoin que d'une seule instance de *Field*. Grâce à ce patron de conception, on s'assure que l'application ne produira qu'une seule et unique instance de *Field*. Il s'agit alors du point d'accès global pour les autres objets.

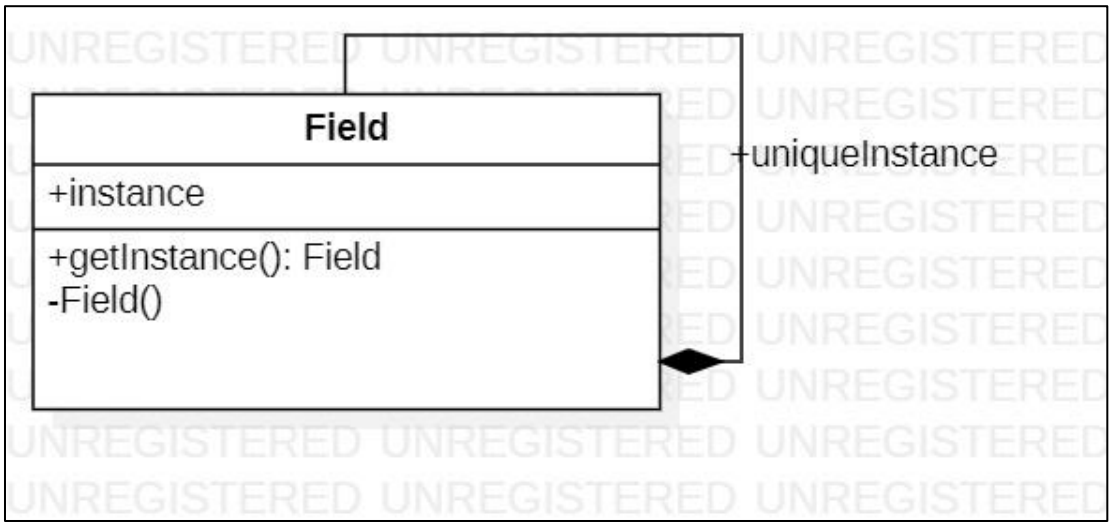

Figure 2: Diagramme du Singleton

III- L’architecture MVC

Pour le MVC nous avions commencé à séparer les classes de sorte à avoir pour chaque classe une vue et un contrôleur. Dans la vue, on cherchait à mettre tout ce qui est animation c'est-à-dire affichage des joueurs, de la balle, du terrain... Et dans le contrôleur, tout ce qui concernait les actions des joueurs, c’est à dire les déplacements, les shoots... . Malheureusement cela n’a pas fonctionné donc on a décidé d’abandonner ce design pattern par manque de temps. Pour autant, on a quand même des vues et contrôleurs dans certaines classes.

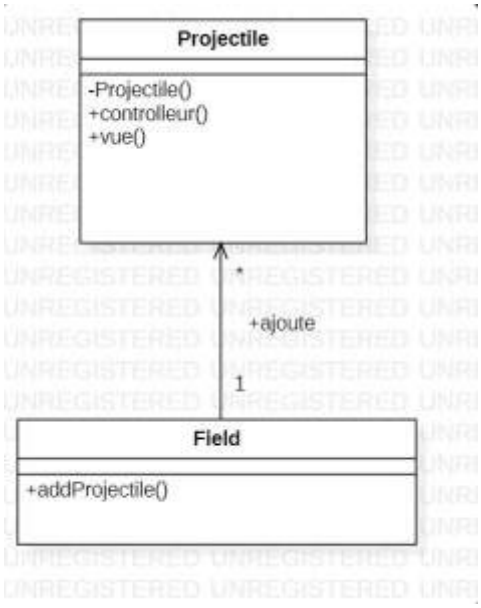

Figure 3: Diagramme MVC

Conclusion

Pour conclure, lors de ce projet nous avions comme objectif de réaliser ce jeu en y apportant des améliorations telles que : l’ajout d’un score, d’un timer pour la durée d’emprisonnement d’un joueur (si un joueur touche un joueur de l’équipe adverse alors il devient prisonnier/inactif pour une durée de 30 secondes), des obstacles de sorte que la balle ne puisse pas atteindre sa cible, la collision entre balle et joueur et entre balle et murs...

Mais par manque de temps et par la difficulté de certaines tâches on a réalisé uniquement : la gestion des tirs, la collision mur-joueur, le déplacement des joueurs humains et la création d’autres joueurs sur le terrain ainsi que l’implémentation de deux design patterns.

Pour le MVC on a essayé de faire le diagramme même s’il n’est pas complet.

Ce projet nous a permis de nous familiariser avec javafx et maven mais également améliorer nos compétences en java.